"Inteligentní" firewall

Petr Halický, Josef Horálek
Faculty of Informatics and Management
University of Hradec Kralove,
Hradec Kralove, Czech Republic
p.halicky@seznam.cz, josef.horalek@uhk.cz

ABSTRACT

V dnešní době se potřeba ochrany dat ukazuje stále naléhavější. První linii obrany by měl představovat firewall; síto filtrující komunikaci přes síťové rozhraní počítače. Právě připojení k internetu je tím nejzranitelnějším přístupovým bodem. Každé řešení však časem bude překonáno. Udržovat aktuální nastavení je mnohdy časově náročné a nezřídka zcela vyloučené. V této práci je navrženo řešení využívající určité "autonomní" algoritmy jak pro samotné úvodní nastavení a inicializaci firewallové aplikace tak i pro vytváření pravidel filtrování na základě analýzy probíhající komunikace a jejího porovnání s platným seznamem norem. Navržené řešení se v rámci testovacího chodu osvědčilo a úspěšně prokázalo svoje schopnosti automatického nastavení a tvorby pravidel. Aplikace dokáže značně ušetřit čas při nastavování a správě firewall na OS GNU/Linux.

Keywords-firewall; adaptable; network security; OS security; packet-filtering

I. ÚVOD

Bezpečnost. Soukromí. Ochrana dat. V dnešní moderní době jsou to stále častěji diskutovaná témata. Jak lze bezpečně ochránit údaje a data udržovaná v digitální podobě. Ale nejedná se pouze o soubory, v ohrožení mohou být i další komodity. Útoky zaměřené na počítače nebo na důležité uzly v počítačových sítích mohou vyústit v obrovské škody i na samotných zařízeních, nemluvě o ztrátě nezálohovaných informacích a času, který bude nutný jak pro nápravu problému tak jejich opětovné vytvoření.

Většině těchto problémů se přitom dá předcházet. V dnešní době, kdy výkon jediného procesoru pokryje potřeby i několika náročných aplikací, není přece problém obětovat jistou část výpočetního času pro zajištění ochrany. Antivirová řešení už v mnohých situacích dokázali, že jejich přítomnost v systému může opravdu předcházet vážným ztrátám a narušením systému. Navíc díky masivnímu a dnes již prakticky všudypřítomnému připojení k internetu není problém mít antivirové řešení vždy s nejaktuálnějšími informace pro hledání narušitelů, tak zvolit méně náročnou aplikaci ve formě externích virových skenerů.

Jenomže, i ten nejvýkonnější antivirový program funguje až po přijmutí vadných dat a jen zlomek z těchto aplikací dokáže reagovat i na jiné než jen virové hrozby. Často je tedy nutné doplnit i druhou vrstvu ochrany. Tou jsou stále dokonalejší firewallová řešení.

Řádně fungující firewall dokáže filtrovat příchozí komunikaci a tím snížit množství dat, která musí být následně zkontrolována antivirovým řešením. V důsledku tak nejen že zvyšuje zabezpečení počítače, v případě místního firewallu, ale dokáže samozřejmě ochránit i celé intranetové sítě, při použití dedikovaného hardwaru.

Kvůli nárůstu bezpečnostních průniků využívajících více rozličných metod, DoS (Denial of Service ~ odepření služby) a viry, je stále těžší udržet funkční ochranu. Je navíc nutné kombinovat více rozličných úrovní a metod pro zabránění nebo minimalizování útočníkových možností a případného dopadu. Objevují se tedy pokusy o vložení schopnosti rozpoznání takovýchto pokusů o přístup přímo do firewallového řešení pomocí metodologií AI pro rozpoznání závadného nebo podezřelého obsahu nebo podezřelého síťového provozu [1].

Mnoho typů útoku je však vedeno ve více vlnách, z nichž každá má svůj specifický úkol. Nezřídka tak před samotným úderem na vnitřní struktury může dojít k vyřazení nebo ochromení obranných mechanismů. I jednodušší firewally je možné zahltit a pokud není dostatečně odolný, dojde k otevření vnitřní sítě po jeho pádu. Je tedy nezbytně nutné dokázat včas odhalit kolabující řešení popřípadě běžící na hranici svých možností; a to nejen při útoku [2].

Dalším faktorem jsou například veřejně dostupné webové aplikace, které počítají s možností vysokého počtu připojených uživatelů a jen obtížně se dají chránit, aniž by zároveň nedošlo k poklesu výkonu. Většina standardních metod navíc používá ochranu založenou na porovnávání vzorů, která však selhává u tzv. "zero-day-attack", pro něž neexistuje odpovídající vzor. Variantou může být zavedení mechanismů, které jsou schopny tyto útoky odhalit bez závislosti na vzoru pro porovnání [3].

Nezřídka však stačí vytvořit a nastavit firewall a nechat ho kontrolovat probíhající komunikaci. Takovéto řešení, ač se zprvu může zdát být více než vhodné a dostačující, se brzy může ukázat jako zdroj mnoha komplikací, například s růstem objemu přenášených dat může při provádění kontroly docházet k velkému zpomalení toku. Není třeba dodávat, že navíc s každým rozšířením chráněné oblasti je navíc nutné

přizpůsobit pravidla. Každý zásah však zvyšuje režii, kterou firewall při kontrole vytváří. Variantou může být využití technik pro třídění používaných pravidel [4] na základě analýzy četnosti využití.

Je tedy nutné najít rovnováhu mezi možnými řešeními, která by zajistila potřeby rostoucí sítě, přibývajících hrozeb, s co nejmenším dopadem na výkon síťového přenosu a současně nepřinesla neúměrné náklady na svoje zprovoznění a následný provoz. Zároveň by nástroj měl být schopen automatického přizpůsobení a pokud možno nabízet přívětivé prostředí pro uživatele.

Tato práce vznikla na podkladu mé rozpracované diplomové práce "Problematika a optimalizace firewall nad OS Linux", která bude obhajována v létě 2015.

Většina této práce byla napsána za pomoci nástroje NEWTON Dictate 4 [5].

II. DEFINICE PROBLÉMU

Zabezpečení interních dat není jednoduché. Abychom měli jistotu, že se do vnitřní sítě nemůže dostat žádný nepovolaný útočník je potřeba vystavět robustní ochranu, která nemůže spoléhat na jediné řešení. Nezřídka se tedy stává, že není použito jediného firewallového návrhu, ale dochází ke kombinování několika samostatných celků v kaskádovité struktuře.

Základním rozdělením firewallových řešení je jejich umístění v rámci topologie sítě. První skupinu tvoří samostatná hardwarová zařízení, která se zpravidla umisťují mezi vnější směrovač a zbytek sítě. Toto umístění může fungovat dvěma způsoby; první možnosti je přímé vložení do cesty datového toku, kdy získá bezprostřední přístup k veškerému provozu a umožňuje okamžité uzavření cesty v případě nalezení podezřelého obsahu. Druhou možností nasazení tohoto typu zařízení je duplikace veškeré komunikace pro analýzu. V tomto případě je možné použít důmyslnější metody, které by za normálních okolností značně zatěžovaly sít a snižovaly celkovou propustnost. Nevýhodou je samozřejmě skutečnost, že jakákoliv zjištění nemohou být okamžitě aplikována, protože analýza probíhá s časovým odstupem vzhledem k zachycené události.

Druhou skupinu firewallových řešení jsou softwarové programy umístěné povětšinou přímo na klientském počítači, koncové stanici. Takovéto aplikace nezřídka dokážou zastoupit samostatné, dedikované řešení v podobě specifického zařízení, jsou omezeny pouze na jediný stroj. Tento fakt nechává, i v případě úspěšně odraženého útoku, volnou cestu k napadení dalších počítačů v rámci interní sítě, které však nebyly v původním zmařeném útoku nijak postiženy; jejich firewallové řešení nemusí být dostatečně připraveno, protože softwarové aplikace své informace v rámci místní sítě jen zřídka sdílejí. Existují sice řešení, která odesílají statistická data a zjištěné informace na centrální server, kde dochází k jejich analyzování, vytvoření informačních zpráv a jejich následné rozeslání do celé sítě. Toto řešení však počítá s existencí takového serveru nebo se zakoupením specifického softwaru, který má tuto podporu od svého vydavatele. I tak zde však existuje prodleva mezi původní informací a jejím následném rozeslání.

Není zajisté nutné dodávat, že oba tyto přístupy se dále vnitřně dělí podle principů na jakém dané řešení dokáže fungovat. Tím je myšlena hlavně metodologie a algoritmy, které jsou používány k analýze a definování nežádoucí komunikace a nástroje sloužící k odhalování či prevenci různých druhů útoků.

Je vhodné kombinovat jak různé typy přístupu; použít samostatné zařízení v kombinaci s aplikačním softwarem, tak jednotlivé metodologie a typy zařízení, která by měla být umístěna v pořadí, v jakém bude docházet k maximálnímu omezení datového toku nežádoucí komunikace a tedy snížení náročnosti filtrování, analyzování náročnějšími druhy ochranných aplikací.

Ani tato kombinace však nemusí nutně znamenat stoprocentní zabezpečení interní sítě. Podle testování laboratoří NSS velké skupiny firewallových řešení se ukázalo, že ani přední výrobci samostatných zařízení nejsou zcela odolní proti chybám a moderním útokům. Podle vyjádření byly všechny testované systémy náchylné na útok, který byl veden přímo proti těmto řešením, v případě používání jejich původního nastavení. Odstranění této chyby je samozřejmě možné změnou konfigurace zařízení, což však může mít negativní dopad na výkon [6].

Variantou se tedy mohou ukázat softwarová řešení využívající nejrůznější principy pro analyzování a identifikování závadného obsahu. Jedním takovým přístupem může být využití tokenů, zástupných objektů, definujících obsah hlavičkového souboru, který bude následně předán k analýze a zavedení příslušné akce. Výhodou řešení, a hlavně jeho založení na analýze i chybějících informací vzhledem k statistické průchodnosti dat, je vysoká schopnost odhalit i doposud neznámé vzory, jak počítačových virů tak druhů útoků, které by mohly být použity. Takovéto řešení by mělo schopnost nejen ochránit vnitřní síť, ale i, při správném nastavení a dostatečných datech, opravit chyby v probíhající komunikaci a tím eliminovat hrozbu bez nutnosti omezení spojení s vnějším světem. Primárně je však navržen pro použití s webovými aplikacemi, u kterých lze celkem jednoduše předvídat druh komunikace s klientem. Není vhodné nasazení pro ochranu počítačového systému před hrozbami z vnější sítě, které je v rámci tohoto dokumentu hledáno [3].

Většina ochranných řešení pracuje na omezené vrstvě síťového protokolu, a proto může být vcelku jednoduše zranitelná hrozbou, která se bude vyskytovat mimo kontrolovaný rozsah. Určitým opatřením může být hloubkové inspekce síťového paketu, která bude kontrolovat veškeré informace obsažené v datovém objektu na všech jeho vrstvách. V tomto případě se však více než o uzavřeném řešení hovoří o jakémsi analyzátoru, který by byl umístěn v rámci topologie tak, aby nashromážděné informace předával dále kontrolnímu programu, který by teprve zajistil požadovanou akci, tedy aplikování politiky. Z

tohoto pohledu by se spíše než o jednoduchou aplikaci jednalo o komplexní bezpečnostní systém, který by vyžadoval ke svému správnému působení administrátora, který by byl odpovědný za optimální nastavení a konfiguraci takto komplexního nástroje. Pro použití v menším rozsahu, popřípadě domácí nasazení, by se však jednalo o příliš robustní návrh [7].

Jako použitelnější variantou se jeví využití paketového filtrování, což je technologie, která se ve velké míře využívá i v rámci jiných řešení. V základu se jedná o kontrolu na úrovni paketů a jejich porovnání s daným vzorem, který rozhodne o zahození respektive propuštění dále v jeho cestě. Není však možné vytvořit jednorázovou konfiguraci, která by následně dokázala odolat veškerému nežádoucímu přístupu. Důsledkem je tedy vyžadována neustálá kontrola nebo aktualizace záznamů, pravidel. V tomto případě by však s přibývajícím počtem zásahů docházelo k neúměrnému nárůstu počtu pravidel a tedy nevyhovujícímu nárůstu nepřehlednosti. Řešením se může zdát využití algoritmů pro dynamické setřídění a optimalizaci těchto pravidel na základě analýzy jejich použití, aplikování na příchozí a odchozí komunikaci. Jako samostatné řešení je toto však zcela nepoužitelné, jelikož vyžaduje existenci pravidel a není samostatně schopno jejich generování na základě probíhající komunikace [4].

Ze zde představených řešení a teoretického úvodu do principu problému jasně vyplývá potřeba specifičtěji zaměřeného řešení, které by vyhovovalo bezpečnostním výkonem i uživatelskou přívětivostí a pokud možno podporovalo škálovatelnost ochrany od rozsáhlé firemní sítě až po použití na jednotlivých zařízení i v rámci jedné domácnosti.

III. NOVÉ ŘEŠENÍ

Vzhledem ke skutečnosti, že v předchozí kapitole popsaná řešení plně nepokryla problém, je nutno vyhledat, navrhnout nové řešení, které bude s největší pravděpodobností mnohem úspěšnější v oblasti vymezené daným problémem. V rámci návrhu nového přístupu se zaměříme převážně na problematiku firewallového řešení, které lze aplikovat na operačních systémech typu UNIX a GNU/Linux.

Tyto operační systémy jsou méně rozšířené než nejznámější operační systém z dílny společnosti Microsoft, ale i tak jsou vystaveny možnému riziku při komunikaci s vnějším světem prostřednictvím síťového rozhraní, ačkoliv ne v takové míře jako operační systémy Windows.

Vzhledem k povaze operačních systémů Linux a již obsaženému řešení na bázi paketového filtru, iptables, bude nově navržené řešení využívat tuto interní funkcionalitu jako prostředek přímého vynucování pravidel, která budou vznikat na základě vlastní činnosti aplikace.

Kvůli zajištění určité funkcionality na všech distribucích se nabízí pro řešení daného problému dva programovací jazyky. První variantu představuje implementační jazyk operačního systému a sice C. Představuje možnost přímého přístupu jak k funkcionalitám jádra systému tak k souborům hardwaru a jejich ovladačům. Jako druhá možnost se nabízí velice populární a silně rozšířený programovací jazyk založený na objektovém přístupu JAVA. Obecně se traduje, že tento jazyk není ve srovnání s ostatními nijak výrazně rychlý a nelze v něm úspěšně naprogramovat firewall, který by byl schopen funkčnosti.

Na základě zkoumání možností implementace v obou těchto jazycích a testování dočasných řešení, bylo nakonec rozhodnuto implementovat aplikaci na základě objektového přístupu, který se ukázal být více než adekvátním v porovnání s druhým přístupem. Toto rozhodnutí však přineslo drobnou potíž spojenou s nutností existence překladové knihovny, která by zajistila přístup k nižším funkcionalitám, síťovému rozhraní, z oblasti aplikačního softwaru, za kterých lze považovat virtuální stroj, ve kterém běží všechny aplikace napsané v jazyku JAVA. Hledání řešení tohoto problému přineslo několik možností, jak velice efektivní a rychle vyřešit spojení na nižší vrstvy, přičemž byla vybrána varianta pro použití překladové knihovny částečně napsané v obou zvažovaných programovacích jazycích.

Vzhledem k požadavku obsluhy celé aplikace z přívětivého uživatelského prostředí je nutno separovat veškerou logiku do samostatných vláken, které by při své činnosti nezatěžovaly uživatelské rozhraní a umožnili tak plynulé fungování, jak automatické detekce tak uživatelského přizpůsobení a kontroly funkčnosti programu. Analýza problematiky a návrhu řešení přinesla i rozdělení a doporučený počet těchto vláken. Vzhledem k náročnosti a množství paketů, které se očekávají procházet síťovým rozhraním a které musí tato aplikace být schopna zkontrolovat, je první vlákno určeno výhradně k zachytávání a přepsání obsahu paketů do interní struktury vhodné pro další analýzu a zpracování, podobné s řešením [3]. Tato struktura by měla být navržena, aby podporovala jak hlavní strukturu paketu tak následnou strukturu pravidel, která mohou být dále implementována a vytvářet samostatné firewallové rozhraní.

Vzhledem k použité knihovně pro překlad funkcionality a režimu přístupu k zachycení paketů, není ani tak důležitá strukturovanost paketů, hierarchické propojení hlaviček a obsahů, jako spíš celkové vzezření a obsah informací v daném paketu. Podobně jako [7] i navržené řešení nepracuje pouze na základních vrstvách ale dokáže analyzovat veškerý obsah paketů v rámci celého ISO/OSI modelu. Jelikož kontrola nákladu každého paketu nemá velký význam pokud by nebyl připojen i virových skener nebo zavedeny algoritmy pro detekci abnormálních kódů, je tato činnost vyhrazena pouze pro přenos aktualizačních dat mezi jednotlivými stanicemi, které budou používat tuto aplikaci v rámci jedné sítě.

V rámci procesu zachycení a přepisu paketů bude docházet k rozlišení, zda byl zachycen datový paket, který

nadále bude směrován pro kontrolu, nebo aktualizační, který bude předán pro okamžité zpracování bez další kontroly. Obě tyto aktivity budou obslouženy v samostatných vláknech, aby se zajistila maximální průchodnost síťové komunikace a maximalizoval počet zachycených paketů pro kontrolu.

V rámci analýzy budou kontrolována nejen data obsažená v objektu, který reprezentuje paket, ale i počet výskytů identického objektu za uplynulou časovou jednotku; tuto jednotku si bude moci uživatel libovolně definovat, avšak už při prvním spuštění si aplikaci dokáže nastavit podle svých interních ukazatelů vlastní interval. Přestože se jedná o velice jednoduchý výpočet, použitá metodika, byť nijak komplexní, se podobá navrženému řešení [8]. Pravidlo, které bude výsledkem zahrnutí paketu do filtrovacího seznamu, bude podrobeno hlubší analýze pouze v případě, že se dané pravidlo již ve výsledném seznamu nevyskytuje. Pokud tomu tak bude, je objekt reprezentující paket a svým způsobem reprezentující i výsledné pravidlo předán analytické třídě, která provede jeho analýzu. Analýza bude tvořena několika kroky, a sice kontrola zdroje / cíle vzhledem k seznamu povolených, respektive zakázaných adres, které si uživatel může nadefinovat pomocí grafického rozhraní. Následně proběhne ověření původu paketu, a to skrze MAC adresu, která by měla ukazovat na nejbližší prvek v síťové hierarchii. Pokud by neodpovídala, lze předpokládat, že paket byl podvržen a tudíž mu nelze důvěřovat; bude označen za závadný. Následně proběhne porovnání čísla portu a protokolu vzhledem k seznamu adekvátních dvojic pro hledání odpovídajícího páru. Tato kontrola proběhne ve dvou stupních v obou směrech daného páru, aby se vyloučila možnost chybně interpretovaného výsledků. Seznam těchto dvojic, pravidel přiřazení portů a služby, je opět možno v plné míře editovat na základě grafického rozhraní. Pokud by byl paket v kterékoliv fázi analýzy označen za neplatný, bude do jeho struktury uložen popis, z jakého důvodu se tak stalo, který bude uživateli přístupný ze seznamu všech aktivních pravidel, která opět může libovolně upravovat vytvářet či vymazávat pomocí uživatelského rozhraní.

Bude-li povolena funkcionalita odesílání aktualizací bude nové pravidlo, určené na základě analýzy pro přidání do seznamu aktivních, odesláno do dalšího vlákna, které odpovídá za jeho přepsání do vlastního paketu a odeslání skrze síťové rozhraní.

Mimo této hlavní funkcionality bude aplikace dále schopna správy uživatelských profilů, které mohou mít na sobě navzájem nezávislá pravidla, čímž bude umožněna přenositelnost aplikace mezi jednotlivými pracovními stroji, například ve formě flash disku, ale současně i provázání jednotlivých profilů pomocí sdílení pravidel z aktivního do seznamů ostatních pravidel, bude-li tato funkcionalita nastavena uživatelem. V případě vytváření nového profilu bude aplikace schopna inicializace všech nastavení na základě zdrojového profilu, který nebude možné skrze rozhraní nijak změnit.

Vzhledem k vysoké automatizaci celé aplikace bude již při prvním spuštění vše nezbytné automaticky odvozeno a nastaveno bez nutnosti zásahu uživatele s výjimkou jediného bodu; určení aktivního síťového rozhraní, na které se má aplikace zaměřit při kontrole.

IV. Implementace řešení

Jak již bylo zmíněno v předchozí kapitole pro vývoj aplikace byl vybrán objektový programovací jazyk JAVA.

Velká část aplikace je programována výhradně za účelem podpory uživatelského rozhraní a tedy nemá přímý dopad na hlavní funkcionalitu aplikace. Z toho důvodu nebudou tyto části blíže v této kapitole popsány, jelikož se jedná o podpůrné struktury. V podobné míře nebudou zahrnuty ani entity určené k interní správě dat.

Zřejmě nejdůležitější částí při tvorby aplikace bylo rozhraní, určené pro zachytávání a přepis paketů z daného síťového rozhraní do interní struktury pro další použití v rámci aplikace. Tento objekt je založen na volně dostupné knihovně jNetPcap [9]. Pro svoji funkcionalitu vyžaduje tato třída přečíst hodnoty z nastavení aplikace, aby mohla správně reagovat v případě zachycení paketu, který nese aktualizační informace. Tato třída je odpovědná za identifikaci zdroje a cíle paketů na základě MAC adres, IP adres, čísla portu, přenosové technologii, a pokud paket obsahuje i informaci o službě.

```
...
Ethernet ethernet = new Ethernet();
if(packet.hasHeader}ethernet)){
          byte[] macDes = ethernet.destination();
          byte[] macSou = ethernet.source();
}
Ip4 ip = new Ip4();
if(packet.hasHeader(ip)){
          destination = FortmatUtils.ip(ip.destination());
          source = FormatUtils.ip(ip.source());
}
...
```

Kvůli správné funkčnosti síťové komunikace a identifikaci směru paketu, kterou sám nedokáže sdělit, hlavní třída aplikace udržuje informaci o propojeném síťovém rozhraní včetně rozsahu adres sítě, do které zařízení spadá. Tyto údaje slouží pro identifikaci směru na základě zdrojové a cílové adresy, ale i pro správnou funkcionalitu odesílání aktualizační paketů.

Odpovědné vlákno za samotné zachycení paketů inicializuje spojení pomocí rozhraní knihovny a následně čeká, dokud není zachycena zpráva procházející tímto rozhraním v libovolném směru. Jakmile se tak stane, dojde k přečtení paketu, jak bylo popsáno v předchozích odstavcích, a následnému přepsání obsahu do interní struktury pro další využití. Díky informacím udržovaný v rámci hlavní třídy, a tedy hlavní paměti celé aplikace, dojde k určení dodatečných vlastností paketu jako jeho směr či typ vstupního, respektive výstupního rozhraní. Toto vlákno dále rozlišuje, zda se jedná o zachycený paket běžné komunikace nebo aktualizační zprávu, která je bez další analýzy předána vláknu odpovědnému za aplikování těchto pravidel. Vzhledem k

implementování jednoduchých statistických výpočtů [8] dochází k uložení paketu ve třech kopiích pro inkrementační analýzu množství identické komunikaci procházející rozhraním.

```
...
Pcap pcap = Pcap.openLive(nic.getName(), 64 * 1024,
                Pcap.MODE_PROMISCUOUS, 1000, errorBuilder);
if(pcap == null){
      runScan = false;
      setOutput("pcap is null");
} else {
      Pcap.loop(1, jph, "");
      ...
```

Dalším vláknem podle funkcionality je „vymahatel“, který čeká, dokud mu není předložen seznam pravidel pro následnou analýzu. Analýza paketů byla oddělena od jejich zachycení kvůli náročnosti způsobující nadměrné zpomalení komunikace a dále kvůli nutnosti sběru statistických dat v daném časovém intervalu. Toto vlákno prověří, zda se pravidlo už nevyskytuje v seznamu aplikovaných, což může nastat vzhledem k možnosti překročení nebo porušení pravidel při prvním výskytu, který je nadále zkopírován v minimálně dalších dvou instancích, než je komunikace zakázána. Následně je otisk paketu podroben analýze pomocí analytické třídy, která vrací jednoduchou informaci, zda má nebo nemá být zahrnut do aktivních pravidel. Pokud je otisk takto označeno, dojde k jeho aplikování, jak do seznamu aktivních pravidel, která jsou dynamický udržována v paměti pro kontrolu, do seznamu pravidel v rámci samotného rozhraní firewall, tak do záložního seznamu daného uživatelského profilu, respektive všech profilů, které jsou s tímto svázány v rámci sdílení pravidel. Pokud je nastaveno odesílání aktualizačních paketů bude takováto zpráva sestavena a odeslána; o tuto činnost se stará samostatné vlákno vzhledem k náročnosti vytvoření zprávy a ne zcela úspěšnému odeslání kvůli problematickému přístupu k rozhraní.

```
...
if(newRule && Analyzer.performAnalyze(
                rule.setDimension(dimension),
                whoami,
                properties,
                serviceAndPort)){
      appliesRule.add(
                rule.setDescription(
                          Analyzer.getReason()).toString());
      saveRule();
      ...
```

Oproti teoretickému návrhu došlo k vytvoření dodatečného vlákna odpovědného za přepínání kontextu uživatele, respektive jednotlivých profilů, a současně otáčení seznamu pravidel pro analýzu na základě uplynutí daného časového intervalu. Přestože celá aplikace je tvořena jako asynchronní záležitost, i z důvodu ne zcela přívětivého prostředí pro synchronizaci jednotlivých vláken, je toto vlákno jako jediné schopné přerušit činnost analytického vlákna a to v okamžiku přepínání kontextu.

```
...
if(calendar.before(Calendar.getInstance())){
      setCalendar(Calendar.getInstance());
      ipTablesRulesIndex.add(0,
                ipTablesRulesIndex.remove(
                ipTablesRulesIndex.size() - 1));
} else {
      ...
```

Následující vlákno bylo vytvořeno za účelem propagace zachycených aktualizační zpráv do vlastních souborů a objektů v místní paměti počítače. Vlákno nejprve zkontroluje existenci pravidla v rámci seznamu aplikovaných a pokud nenajde shodu, zanese zachycený paket, respektive jím nesenou informaci v podobě nákladu, do lokálních záznamů.

```
...
if(performUpdate){
      bridge.forwardCommand(rule.getAppendRule());
      appliesRules.add(rule.toString());
      saveRule(rule);
}
...
```

Posledním vláknem je „odesílač“ odpovědný za tvorbu funkčního předpisu paketu a jeho odeslání skrze aktivní síťové rozhraní. Vzhledem ke skutečnosti, že přistupuje ke stejnému rozhraní jako vlákno pro zachytávání paketů, dochází k určité neúspěšnosti přenosu. Toto zjištění bylo částečně kompenzováno případným zařazením vytvořené zprávy pro opětovné odeslání po uplynutí čekacího intervalu. Počet pokusů před zahozením zprávy je možné nastavit skrze uživatelské rozhraní.

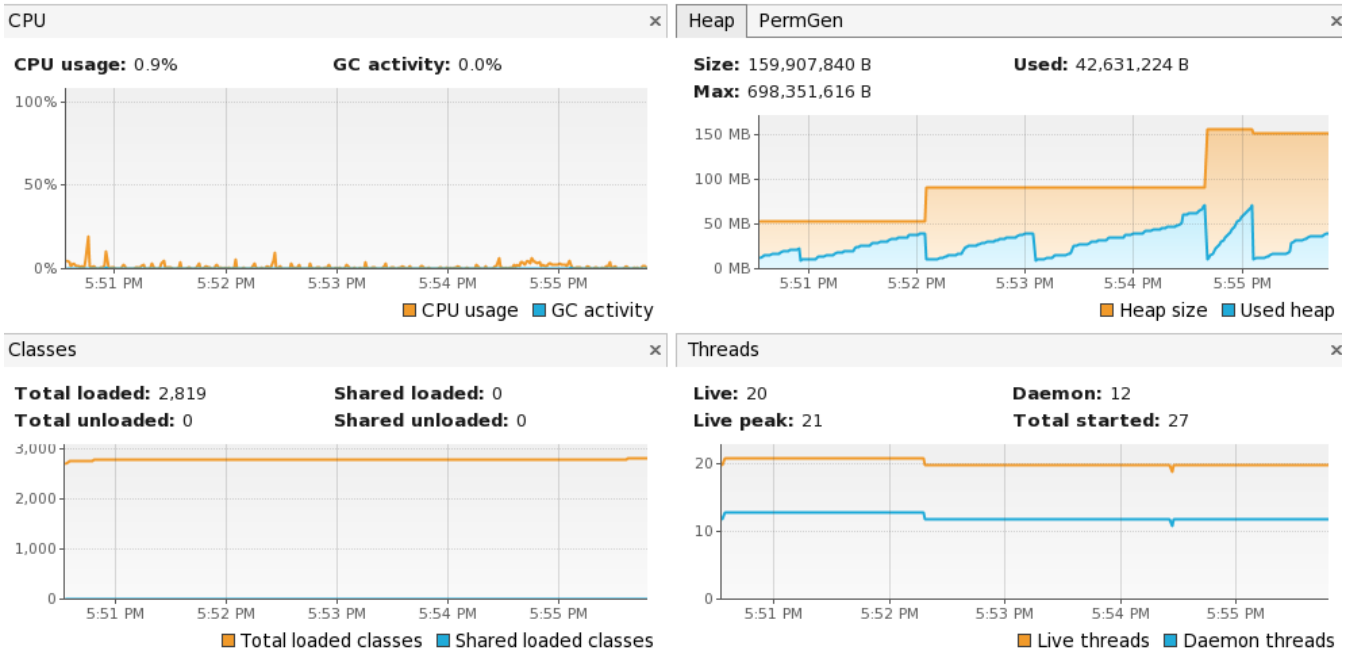
```
...
String message = rule.setWhoami("netUpdate").toString();
JPacket packet = constructIpPayloadPacket(message);
if(packet != null && pcap.sendPacket(
                ByteBuffer.wrap(
                packet.getByteArraz(0, packet.size()))) == 0){
      ...
```

Veškeré nastavení aplikace je spravováno skrze jednotné rozhraní uložené v jediném souboru typu properties. Oproti základnímu rozhraní, které programovací jazyk nabízí byla vytvořena obalová třída zapouzdřující některé tyto metody a doplňující funkcionalitu na vyšší úrovni. V základu obsahuje soubor ukázkový profil, který je použit při úvodním nastavení aplikace, což umožňuje zcela automatické uvedení do provozu bez nutnosti zásahu uživatele. Jsou-li vyžadovány nějaké dodatečné informace, aplikace je schopna si je autonomně obstarat v okamžiku svého startu nebo v průběhu několika mála okamžiků po něm z dat, která si umí shromáždit. Jedinou výjimku představuje identifikace aktivního síťového rozhraní, které má být touto aplikací střeženo. Tato funkcionalita vyžaduje zásah uživatele a při úvodním spuštění může vést až k ukončení aplikace, pokud uživatel neoznačí platné rozhraní. Aplikace identifikuje všechna dostupná rozhraní v rámci počítače a dotáže se uživatelé na označení aktivního; není vyžadována větší spolupráce pokud uživatel sám nepožaduje. Properties soubor obsahuje, jak globální nastavení zahrnující všechny profily, tak lokální informace pro jednotlivé profily, které jsou dynamicky zanášeny do tohoto souboru. Vzhledem k očekávanému počtu pravidel pod každým uživatelským profilem jsou tato pravidla udržována v samostatných souborech, které jsou dynamicky vytvořeny a spravovány samotnou aplikací.

Stěžejní funkcionalitu, rozhodnutí o osudu paketů na základě analýzy informací, představuje analytická třída. Pracuje na principu porovnání údajů s dynamicky generovanými a upravovaný vzory. Nejdříve je kontrolována příslušnost zdrojové / cílové adresy do rozsahu povolených / zakázaných. Následně je ověřována posloupnost údajů identifikujících cestu skrze síť a sice na základě identifikované MAC adresy nejbližšího síťového prvku a zdrojové MAC adresy uchované v paketu. Pokud nesouhlasí je pravděpodobnost, že paket byl podvržen. Následně

V. TESTOVÁNÍ VYVINUTÉ APLIKACE - ŘEŠENÍ

Protože aplikace není zcela hotova, bylo by minimálně bezúčelné provádět plnohodnotné testování jejího výkonu jakožto firewallového řešení a porovnávat ho s dalšími podobnými aplikacemi. Navzdory tomu však lze aplikaci spustit v testovacím režimu a kontrolovat její funkcionalitu alespoň na základní úrovni bez výstupu ve formě benchmark testů či číselného hodnocení, které by bylo schopné srovnání. Zároveň kvůli omezeným prostředkům nelze úplně otestovat



Obrázek 1Vytížení procesoru, využití paměti, načtené objekty a spuštěná vlákna, VirtualVM

dochází ke kontrole statistického údaje a sice souhrnného počtu identických paketů, které daným síťovým rozhraním prošli v rámci časového intervalu. Poslední kontrola je vícestupňová a představuje ověření služby a čísla portu. Kontrola probíhá dvěma směry kvůli maximálnímu vyloučení možné chyby. V prvním kroku je kontrolováno číslo portu podle definované služby, aby se následně kontrolovala služba podle uvedeného čísla portů. Alespoň jeden z těchto kroků musí skončit kladným výsledkem, jinak je paket označen za závadný.

O výsledku této analýzy je uživatel informován prostřednictvím sdělení připojeného k pravidlu.

```
...
List<String> tempWhite = properties.getProperties(
                whoami + ".whitelist");
for(String s:tempWhite)
      if(partOfAddress(
                          s, rule.getSource + "/"
                          + rule.getMask()) == 1)
                return false;
...
```

schopnost aplikace distribuovat a zanášet aktualizační zprávy posílané jinou instancí v rámci stejné sítě.

Veškerý výkon aplikace tak bude odzkoušen v rámci částečného provozu a jako úspěšné otestování aplikace bude považována funkční schopnost generovat pravidla, která budou definovat chování firewallu na operačním systému typu GNU/Linux, schopnost autonomně nastavit vlastní chování po spuštění bez zásahu, nebo jeho nutnosti, uživatelem. Komunikační schopnosti aplikace budou otestovány pomocí falešné druhé verze, vysílací rozhraní na základní úrovni podobné vlastnímu rozhraní implementovanému v rámci samotné aplikace, kde bude odeslán aktualizační paket a očekávaným výsledkem bude jeho zanesení do seznamu aktivních pravidel používaných pro filtrování průchozí komunikace.

Protože aplikace je vytvořena v objektovém programovacím jazyku JAVA, který pro veškeré spouštění využívá virtuální stroj, lze změřit dopad aplikace a výkon jednotlivých vláken pomocí nástroje určeného k vizualizaci spuštěných aplikací v rámci daného virtuálního stroje. Tímto nástrojem se stalo prostředí VisualVM [10].

V momentálním stavu je aplikaci možné nainstalovat jednoduchým rozbalením z archivu, který obsahuje, jak spouštěcí soubor ve formátu *.jar, tak podpůrné soubory

aplikace. Nicméně pro správný chod je nutno, aby v systému byly přítomny knihovny jNetPcap [9]. Vše potřebné pro jejich nainstalování, včetně podpůrného rozhraní, je v archivu obsaženo.

Průběh vytížení procesoru ukazuje drobné výkyvy v okamžiku zvýšené činnosti aplikace. Předpokladem je

síťovém rozhraní byla správně interpretována a obsažené pravidlo bez problémů zaneseno do seznamu. V tomto rozsahu lze tvrdit, že test komunikace byl úspěšný.

Poslední částí principu aplikace, která byla testována, je autonomní schopnost inicializace a nastavení. Tato funkce úspěšně proběhla a umožnila start aplikace, jak dokazují

Threads: 25 **Total CPU Time [ms]:** 32,701

Thread Name	Thread CPU Time [...	Thread CPU Time [ms]
snooper		12,862.177 (0.0%)
RMI TCP Connection(1)-127.0.0.**1**		7,754.748 (0.0%)
AWT-EventQueue-0		3,653.819 (0.0%)
DisposableGC		2,583.32 (0.0%)
AWT-XAWT		1,446.207 (0.0%)
RMI TCP Connection(3)-127.0.0.**1**		1,434.46 (0.0%)
enforcement		611.253 (0.0%)
DestroyJavaVM		589.607 (0.0%)
***** JFluid Monitor thread *****		527.045 (0.0%)
JMX server connection timeout 26		523.373 (0.0%)
rotor		177.827 (0.0%)
***** Profiler Agent Communication Thread**		147.686 (0.0%)
TimerQueue		102.242 (0.0%)
Attach Listener		86.075 (0.0%)
Thread-1		80.731 (0.0%)
Thread-0		55.962 (0.0%)

Obrázek 2Využití procesoru jednotlivými vlákny

aktivita analyzátorů nebo zachycení komunikace v daném rozhraní. Aktivita a množství tříd a vláken, které aplikace spouští a používá, jsou stabilní ve svých původních počtech jen s drobnými výkyvy v obou směrech. Množství alokované paměti vzrostlo v průběhu testu z původních 50MB na 150MB, ačkoliv množství reálně využívané paměti kolísá podle objemu dat, které se udržují ve vyrovnávacích pamětech určených pro přenos informací mezi jednotlivými vlákny, kontrolu dat, a vytváření dočasných proměnných, které slouží k předávání výsledků jednotlivých analýz a záznamů komunikace. Dopad spuštění aplikace na celkový chod systému je minimální, vytížení procesoru v průměru nedosahuje ani 1%. Přestože se zdá, že většina vláken neustále spí, ve skutečnosti pouze čekají na data pro svoji činnost. Podle očekávání největšího výkonu se dožaduje vlákno pro zachytávání komunikace, které představuje srdce celé aplikace.

Z naměřených hodnot je patrné že aplikace nezatěžuje operační systém, přestože aktivně dokáže zasahovat do síťové komunikace a vytvářet pravidla, která jsou následně vynucována na prostředí operačního systému podobným způsobem, jakým činí běžná firewallová řešení.

Test komunikace mezi simulovanými dvěma programy proběhl vytvořením zprávy obsahujícím pravidlo pro přidání do seznamu aktivních, které bylo odesláno z falešného rozhraní. Jelikož komunikace probíhala ze stejného stroje nelze věrohodně tvrdit, že by takováto zpráva byla schopna přenosu skrze síť, ačkoliv konstrukce paketu odpovídá všem standardům a zpráva zachycena při odchozím volání na

výsledky z předchozích testů, které zdokumentovali samotný běh aplikace. V properties souboru se objevily nové záznamy odkazující na lokální nastavení nového uživatelského profilu, který byl vytvořen podle účtu přihlášeného do operačního systému. Současně byl vytvořen původně prázdný soubor pro záznam dat představující pravidla, která zastupují samotné rozhraní firewallu. V průběhu testovacího spuštění byla vytvořena nová pravidla na základě zachycené komunikace, jenž byla aplikována a doplněna o vysvětlení jejich zařazení do aktivního seznamu. Většina těchto paketů (25) porušila pravidla o službě a spojeném portu. Jelikož je testovaný operační systém spouštěn virtuálně, lze množství těchto pravidel vysvětlit snahou virtuálního stroje o komunikaci s počítačem, a přestože je nelze považovat za standardní typ komunikace, dokazuje jejich množství a výsledky, že aplikace dokáže zpracovat i velké objemy průchozích zpráv.

VI. ZÁVĚRY

Z předchozích částí vyplývá, že se úspěšně podařilo navrhnout a implementovat řešení firewallové ochrany, odpovídající definovanému problému.

V rámci testů se prokázaly základní schopnosti aplikace i její celková funkčnost, avšak kvůli stavu aplikace nebylo možné ověřit míru úspěšnosti v porovnání s alternativními produkty a tedy reálně otestovat výkon firewallu.

I přes výše zmíněné však aplikace dokáže značně ušetřit čas při nastavování a správě firewallu na OS Linux. Kromě použití jako firewallového řešení, lze s určitými úpravami nastavení aplikaci použít pro logování síťové aktivity procházející daným rozhraním.

REFERENCE

[1] Mishra, A., Agrawal, A., Ranjan R.: Artificial Intelligent Firewall, In ACAI '11 Proceedings of the International Conference on Advances in Computing and Artificial Intelligence, pp.204-207, 2011, 10.1145/2007052.2007094

[2] Hamelin, M.: Preventing firewall meltdowns, In Network Security, Volume 2010, Issue 6, pp.15-16, 06/2010, 10.1016/S1353-4858(10)70083-0

[3] Krueger, T., Gehl, Ch., Rieck, K., Laskov, P.: TokDoc: A Self-Healing Web Application Firewall, In SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing, pp.1846-1853, 2010, 10.1145/1774088.1774480

[4] Hamed, H., Al-Shaer, E.: Dynamic Rule-ordering Optimization for High-speed Firewall Filtering, In ASIACCS '06 Proceedings of the 2006 ACM Symposium on Information, computer and communications security, pp.332-342, 2006, 10.1145/1128817.1128867

[5] NEWTON Technologies, 15.11.2014 dostupné on-line: http://www.diktovani.cz/

[6] NSS uncovers firewall shotcomings, In Network Security, VOlume 2011, Issue 5, pp.2,19, 05/2011, 10.1016/S1353-4858(11)70045-9

[7] Moon, Ch-S., Kim, S-H.: A Study on the Integrated Security System based Real-time Network Packet Deep Inspection, In International Journal of Security and Its Applications, Volume 8, pp.113-122, 2014, 10.14257/ijsia.2014.8.1.11

[8] Trabelsi, Z., Zhang, L., Zeidan, S.: Firewall Packet Filtering Optimization Using Statistical Traffic Awareness Test, In Information and Communications Security, Volume 7618, pp.81-92, 2012, 10.1007/978-3-642-34129-8_8

[9] jNetPcap, 9.11.2014 dostupné on-line: http://jnetpcap.com/

[10] VisualVM, 15.11.2014 dostupné on-line: http://visualvm.java.net/